IN THE UNITED STATES PATENT AND TRADEMARK OFFICE BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applicant(s):	Jacob Dreyband, *et al.*
Serial No.:	09/844,993
Filed:	April 27, 2001
Title:	DESCRIPTIVE DATA CONSTRUCT MAPPING METHOD AND APPARATUS
Group Art No.:	2166
Examiner:	Srirama T. Channavajjala
Atty. Docket No.:	68156755.5008

Mail Stop-Appeal Briefs
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF UNDER 37 C.F.R. 41.37

A Notice of Appeal was filed in this case on November 2, 2005. Appellant also timely filed an Pre-Appeal Brief Request for Review on November 2, 2005, along with the Notice of Appeal. A Notice of Panel Decision from Pre-Appeal Brief Review was mailed December 5, 2005, indicating that the present Application should proceed to the Appeal. The present Appeal Brief is being filed within three months of the mailing of the Notice of Panel Decision, along with the appropriate two-month extension fee, and thus is being timely filed.

I. REAL PARTY IN INTEREST

The Real Party in Interest in this appeal is Creative Science Systems, Inc. All of the listed inventors executed assignments on April 26, 2001, of all of their respective interests in the present application to Creative Science Systems, Inc.

II. RELATED APPEALS AND INTERFERENCES

There are no known appeals or interferences related to this application at this time.

III. STATUS OF CLAIMS

This is an appeal from the Final Office Action of August 2, 2005, rejecting all of the pending claims. Claims 1-36 were originally filed in present Application. No claims have been withdrawn, canceled or added. Accordingly, claims 1-36 remain pending in the present application and are presented in this appeal. A clean copy of the claims is reproduced in the Appendix.

IV. STATUS OF AMENDMENTS

On July 8, 2003, a first Office Action was mailed rejecting all of claims 1-36. On November 7, 2003, Appellant filed a response to the first Office Action where claims 1-6, 9, 11-13, 15-20, 23-27 and 29-35 were amended. A Final Office Action was mailed on January 15, 2004, rejecting all of pending claims 1-36. Appellant filed a Notice of Appeal on July 15, 2004, in response to this Final Office Action. A Request for Continued Examination was then filed October 14, 2004, along with an Amendment that amended claims 1, 9, 15, 23 and 29. A third Office Action was mailed on December 23, 2004, again rejecting claims 1-36. Appellant filed an Amendment in response to the third Office Action on June 22, 2005, amending claims 1, 9, 15, 23 and 29. Another Final Office Action was mailed on August 2, 2005. Appellant filed a Notice

of Appeal in response to this Final Office Action on November 2, 2005, along with a Pre-Appeal Brief Request for Review.

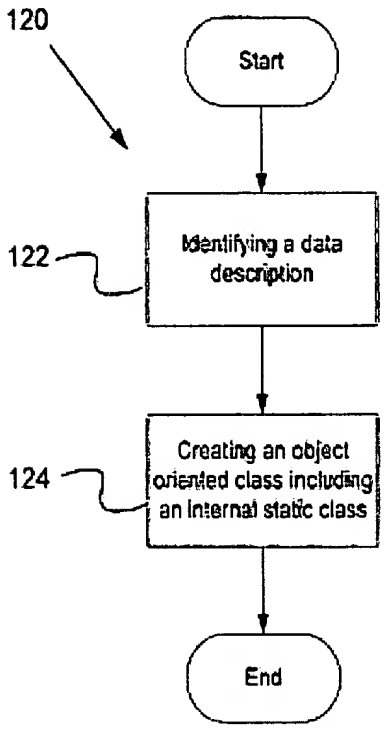
V. SUMMARY OF CLAIMED SUBJECT MATTER

The present invention relates generally to a universally executable object-oriented computer language that is independently executable in any of a plurality of run-time environments. In all instances, the scope of the claims shall be considered on their own merits in light of the specification but should not be constrained by the concise explanation of the subject matter recited in each of the independent claims involved in this appeal. Furthermore, although Appellant has set forth portions of the specification and drawings that support/illustrate the subject matter of the claims, these examples are not exhaustive and shall not necessarily limit the scope of the claims.

A. Concise explanation of subject matter defined in claim 1

Claim 1 provides a method for mapping a descriptive language including a data description having a structure complexity into an object oriented programming language. This exemplary method comprises receiving the data description, and identifying a complex-type element in the data description. In addition, this embodiment of the method includes creating an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to the structure complexity of the data description. Figure 5 of the Application, which is reproduced below, helps to illustrate an embodiment of the claimed method.

Figure 5

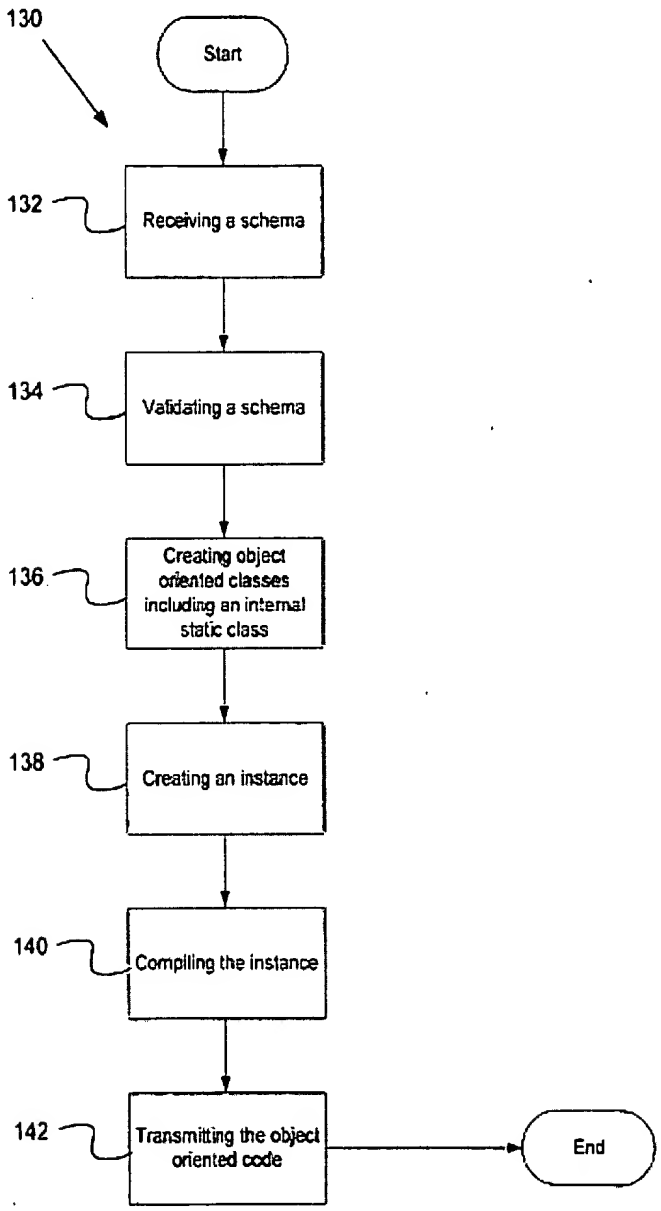


B. Concise explanation of subject matter defined in claim 9

Claimed in claim 9 is a method for mapping a Schema including a structural complexity into an executable object oriented programming language, wherein the object oriented programming language provides a one to one correspondence between the structural complexity of the Schema and the functionality of the object oriented programming language. According to this claim, the method comprises receiving said Schema and validating said Schema. The method also includes creating a set of executable object oriented classes including a set of internal static classes that are independently executable in any of a plurality of run-time environments and to provide a mapping of the Schema into the object oriented language. The method further includes creating an instance corresponding to the object oriented classes, and compiling the instance to provide an object oriented programming code. Figure 6 of the Application, which is reproduced below, helps to illustrate an embodiment of the claimed

method. In addition, Figure 4, also reproduced below, illustrates the mapping of a valid Schema into Java using the claimed method.

Figure 6



90 **Figure 4**

```
92    class person {
94          static class Name extends SimpleTypeString {
96                public static String name() {return "name";}
98                public Name(String v) {super (v);}
          }
100       static class Surname extends SimpleTypeString {
102               public static String name() {return "surname";}
104               public Surname(String v) {super (v);}
          }
106       protected void structure() {
108               addParticle(new SequenceParticle(){
110                     protected void structure() {
112                           addParticle(new ElementParticle("name"));
114                           addParticle(new ElementParticle("surname"));
                      }
                });
          }
    }
```

C. Concise explanation of subject matter defined in claim 15

Claim 15 recites a computer readable medium containing programming which when executed performs receiving a data description, and identifying a complex-type element in the data description. In addition, the medium contains programming which when executed creates an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to a structure complexity of the data description. The above reproduced figures help to illustrate this embodiment of the claimed medium.

D. Concise explanation of subject matter defined in claim 23

Claim 23 recites a computer readable medium containing programming for mapping a

Schema including a structural complexity into an executable object oriented programming language, wherein the object oriented programming language provides a one to one correspondence between the structural complexity of the Schema and the functionality of the object oriented programming language which when executed performs receiving said Schema and validating said Schema. In addition, the medium contains programming that when executed creates a set of executable object oriented classes including a set of internal static classes that are independently executable in any of a plurality of run-time environments and to provide a mapping of the Schema into the object oriented language. Furthermore, when executed the programming creates an instance corresponding to the object oriented classes, and compiles the instance to provide an object oriented programming code. The above reproduced figures help to illustrate this embodiment of the claimed medium.

E. Concise explanation of subject matter defined in claim 29

Claim 29 recites an apparatus for mapping a descriptive language including a data description having a structure complexity into an object oriented programming language. In one embodiment, the apparatus comprises means for receiving the data description, and means for identifying a complex-type element in the data description. In addition, in this embodiment the apparatus comprises means for creating an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to the structure complexity of the data description. The above reproduced figures help to illustrate this embodiment of the claimed medium.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claims 1-3, 8, 9, 14-17, 22, 23, 28, 29 and 36 stand rejected as obvious under 35 U.S.C.

§103(a) over the article entitled, "UML for XML Schema Mapping Specification" by Grady, *et al.*, in view of U.S. Published Application No. 2002/0133811 to Duftler, *et al.* In addition, claims 4-7, 10-13, 18-21, 24-27 and 32-35 stand rejected as obvious under 35 U.S.C. §103(a) over Grady in view of Duftler, and further in view of U.S. Patent No. 6,083,276 to Davidson, *et al.* Because Grady in view of Duftler forms a basis for rejecting each of pending independent claims 1, 9, 15, 23 and 29, the issue presented in this appeal is whether these independent claims are obvious over this combination of Grady and Duftler, if that combination of references is proper. This issue may be resolved if the Board finds that the combination of Grady and Duftler is improper, and/or that this combination does not teach "creating an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to the structure complexity of the data description" as is recited in each of the pending independent claims. As a result, if the Board finds that the combination of Grady and Duftler does not teach all of the elements of independent claims 1, 9, 15, 23 and 29, and their dependent claims, then the Board must find that claims 1-36 distinguish from the cited references and that they should be allowed.

VII. ARGUMENT

Initially, the Appellant respectfully asserts that the Unified Modeling Language (UML) discussed in Grady is not an executable object oriented programming (OOP) language as recited in present independent claims 1, 9, 15, 23 and 29. More specifically, while UML is an object oriented language, it is only a descriptive language and is not an independently executable programming language; hence, Grady's description of UML as an object oriented *design* language. Since UML is a design language (i.e., a descriptive language), it is no more

independently executable than is XML Schema or any other descriptive language. In the programming industry, descriptive languages such as these are used to present data that provides descriptions of certain objects, etc. However, such descriptive code is not by itself executable. In fact, in the industry, once UML code is employed, a separate executable code must then be written to actually execute the descriptions (provided by the UML code) in a run-time environment. Thus, the solely descriptive nature of UML is a limitation.

In contrast, the present claims recite creating an executable object oriented class that is independently executable in a run-time environment and that corresponds to an identified complex-type element, wherein the class includes an internal static class and wherein the internal static class corresponds to the structure complexity of a received data description. Figure 4 of the present Application, reproduced again below, illustrates the claimed code.

90 **Figure 4**

```
92   class person {
94       static class Name extends SimpleTypeString {
96           public static String name() {return "name";}
98           public Name(String v) {super (v);}
       }
100    static class Surname extends SimpleTypeString {
102        public static String name() {return "surname";}
104        public Surname(String v) {super (v);}
       }
106    protected void structure() {
108        addParticle(new SequenceParticle(){
110            protected void structure() {
112                addParticle(new ElementParticle("name"));
114                addParticle(new ElementParticle("surname"));
                }
            });
        }
    }
```

As a result, the code created in accordance with the present claims is a single, independently executable programming language written in a run-time environment that incorporates the characteristics provided by descriptive languages, such as XML Schema. Thus, it does not require separate programming code (as does descriptive code like UML) to actually execute it in a run-time environment.

The recent Final Rejection confirms that Grady does not teach a language having, among other things, an object oriented class that is independently executable in a run-time environment, as recited in independent claims 1, 9, 15, 23 and 29. However, the Final Rejection now cites Duftler for this missing element. In response, the Appellant respectfully asserts that the combination of Duftler with Grady still does not teach or suggest all of the elements of the pending independent claims.

Duftler teaches the use of XML in the Java context to simplify the descriptions of only Java-specific objects. Specifically, Duftler teaches a technique for employing the XML descriptive language as a notational language in order to more efficiently define/describe objects in "JavaBeans" that are used within the Java code. However, the language (and corresponding code) that is generated by the inventions recited in claims 1, 9, 15, 23 and 29 of the present Application still differ from the code taught in Duftler. Specifically, Duftler merely teaches a different means for creating JavaBeans to be executed in Java code such that the JavaBeans do not have to be individually encoded. However, such code of Duftler is still not *independently* executable in a run-time environment. Stated another way, the only executable language discussed in Duftler is Java, which the Appellants have previously demonstrated is not independently executable in any of a plurality of run-time environments. Instead, language in Duftler is only executable in a Java run-time environment. Note that the statement of

".....code to execute script....." in the below example of code from Duftler exemplifies its inability to independently execute the script without additional code.

```
<bsc:component class="foo.Bar">
    <bsc:constructor>
        <bsc:script language= "netrexx">
        ...
        </bsc:script>
    </bsc:constructor>
</bsc:component>
generates the following code:
package foo;
    public class Bar()
    {
        public Bar()
        {
        ..... code to execute script .....
        }
    }
```

In fact, this limitation of Java was one of the motivating factors leading the present Applicants to develop the invention(s) recited in the present claims. Thus, the present claims provide for a universally executable object oriented language, and additionally is not limited to a single language implementation. In contrast, Duftler scripting language still requires "code to execute script," as may be seen through Duftler's disclosure. Since neither Grady nor Duftler teaches or suggests an independently executable object oriented language, as recited in claims 1, 9, 15, 23 and 29, this cited combination of references does not render the pending claims obvious.

Moreover, while Grady teaches the use of XML Schema mapping to the UML code discussed therein, Duftler only discloses the use of straightforward XML to define JavaBeans so that they do not have to be individually coded, and does not even suggest the use of any Schema language in its approach. As a result, there is no motivation for one skilled in the art who is employing a Schema language, such as in Grady and in the present application, to combine the teachings in Duftler since Duftler merely teaches employing XML (a non-Schema language) in a specific descriptive function. As mentioned above, Duftler is teaching the use of XML to

'better' describe an object when creating JavaBeans, which would not be done with a Schema code such as that employed in Grady. There is nothing in Duftler that suggests abandoning the XML descriptive language technique taught therein to try to create JavaBeans with a Schema language, and to do so would likely render the intended purpose of Duftler useless. Moreover, there is nothing in Grady that suggests abandoning the technique for Schema mapping into UML in order to create JavaBeans as taught in Duftler. Thus, in addition to the combination of Grady and Duftler not teaching or suggesting all of the elements in claims 1, 9, 15, 23 and 29, there would be no motivation to combine these two references in the first place. The mere fact that both references deal with innovations in related technology fields is alone not enough to demonstrate a motivation for one skilled in the art to combine the references.

For at least these reasons, independent claims 1, 9, 15, 23 and 29 are not obvious in view of Grady and Duftler. Furthermore, the claims respectively depending from these independent claims also incorporate these distinctions, and are therefore also not obvious in view of Grady and Duftler. Accordingly, Applicants respectfully request that the Board withdraw the §103 rejections based on the combination of Grady and Duftler with respect to the pending claims. Furthermore, claims 4-7, 10-13, 18-21, 24-27 and 32-35 have also been rejected as obvious over Grady in view of Duftler, and further in view of Davidson. However, as discussed above, the pending independent claims are not obvious over the combination of Grady and Duftler. Davidson does nothing to cure these deficiencies in Grady and Duftler, as has not been cited or applied for this reason. Accordingly, the Appellant also respectfully requests that the rejection of these dependent claims also be withdrawn since the independent claims from which these claims depend are not obvious in view of the art of record.

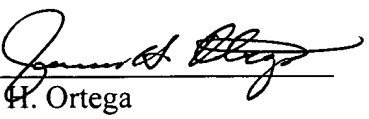
In sum, it has been proven that (1) all of pending claims 1-36 are not obvious over the combination of Grady and Duftler because this combination does not disclose all of the elements of independent claims 1, 9, 15, 23 and 29; and (2) the combination of Grady and Duftler is improper since there is no motivation for one skilled in the art to make the combination. These two references are not directed to solving the same problem, there is nothing in the express teachings in both references leading towards combining, and the knowledge available to one skilled in the art would not lead to a motivation to combine these references.

CONCLUSION

A payment in the amount of $250 is included with the present Appeal Brief. This payment includes the fee required for this appeal (small entity) in accordance with 37 C.F.R. § 41.20(b)(2). In addition, the present Appeal Brief is being filed within three months of the mailing of the Notice of Panel Decision, and thus also includes the appropriate two-month extension fee of $225. If these amounts are not correct, or an overpayment has occurred, then the Commissioner is authorized to deduct or credit Deposit Account No. 13-0480, referencing Attorney Docket No. 68156755.5008.

Respectfully submitted,

Date: February 16, 2006



James H. Ortega
Reg. No. 50,554
BAKER & MCKENZIE LLP
2001 Ross Avenue, Suite 2300
Dallas, Texas 75201
(214) 978-3000 (main)
(214) 978-3099 (facsimile)

VIII. CLAIMS APPENDIX

1. A method for mapping a descriptive language including a data description having a structure complexity into an object oriented programming language, comprising:

receiving the data description;

identifying a complex-type element in the data description; and

creating an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to the structure complexity of the data description.

2. The method as recited in claim 1, wherein receiving the data description comprises receiving an XML Schema.

3. The method as recited in claim 1, further comprising validating the data description.

4. The method as recited in claim 3, wherein said validating further includes using a object finite state machine including a current state to verify a mutator method call against the current state of the object, wherein the data description is invalid when the mutator method call is initiated before the current state is complete.

5. The method as recited in claim 3, wherein said validating includes:

sending a request including said data description to a remote server; and

receiving a validity determination as to said data description.

6. The method as recited in claim 3, wherein said validating includes:

reading said data description into a set of valid descriptor classes; and

creating a set of objects out of the data description wherein the occurrence of an object reflects validity.

7. The method as recited in claim 1, wherein said creating includes a set of object oriented classes selected from the group consisting of: Java, C++ and Smalltalk.

8. The method as recited in claim 1, wherein said creating includes representing a naming space with said internal static class to provide an implementation of said structure complexity.

9. A method for mapping a Schema including a structural complexity into an executable object oriented programming language, wherein the object oriented programming language provides a one to one correspondence between the structural complexity of the Schema and the functionality of the object oriented programming language, the method comprising:

receiving said Schema;

validating said Schema;

creating a set of executable object oriented classes including a set of internal static classes that are independently executable in any of a plurality of run-time environments and to provide a mapping of the Schema into the object oriented language;

creating an instance corresponding to the object oriented classes;

compiling the instance to provide an object oriented programming code.

10. The method as recited in claim 9, wherein said validating includes using a object finite state machine including a current state to verify a function call against the current state of the object, wherein the Schema is invalid when the function call is initiated before the current state is complete.

11. The method as recited in claim 9, wherein said validating includes:

sending a request including said Schema to a remote server; and

receiving a validity determination as to said Schema.

12. The method as recited in claim 9, wherein said validating includes:

reading said Schema into a set of valid descriptor classes; and

creating an instance of a compiler class wherein the compiler class is described in the Schema.

13. The method as recited in claim 9, wherein the set of object oriented classes comprises code selected from the group consisting of: Java, C ++, and Smalltalk.

14. The method as recited in claim 9, wherein said creating an instance includes representing a naming space with the internal static class to provide an implementation of said structural complexity.

15. A computer readable medium containing programming which when executed performs the following:

receiving a data description;

identifying a complex-type element in the data description; and

creating an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to a structure complexity of the data description.

16. The medium as recited in claim 15, wherein receiving the data description comprises receiving an XML Schema.

17. The medium as recited in claim 15, further comprising validating the data description.

18. The medium as recited in claim 17, wherein said validating procedure further includes using an object finite state machine including a current state to verify a mutator method call against the current state of the object, wherein the data description is invalid when the mutator method call is initiated before the current state is complete.

19. The medium as recited in claim 17, wherein said validating procedure includes:

sending a request including said data description to a remote server; and

receiving a validity determination as to said data description.

20. The medium as recited in claim 17, wherein said validating procedure includes:

reading said data description into a set of valid descriptor classes; and

creating a set of objects out of the data description wherein the occurrence of an object reflects validity.

21. The medium as recited in claim 15, wherein said creating procedure includes a set of object oriented classes selected from the group consisting of: Java, C ++ and Smalltalk.

22. The medium as recited in claim 15, wherein said creating procedure includes representing a naming space with said internal static class to provide an implementation of said structure complexity.

23. A computer readable medium containing programming for mapping a Schema including a structural complexity into an executable object oriented programming language, wherein the object oriented programming language provides a one to one correspondence between the structural complexity of the Schema and the functionality of the object oriented programming language which when executed performs the following procedures comprising:

receiving said Schema;

validating said Schema;

creating a set of executable object oriented classes including a set of internal static classes that are independently executable in any of a plurality of run-time environments and to provide a mapping of the Schema into the object oriented language;

creating an instance corresponding to the object oriented classes; and

compiling the instance to provide an object oriented programming code.

24. The medium as recited in claim 23, wherein said validating procedure includes using an object finite state machine including a current state to verify a function call against the current state of the object, wherein the Schema is invalid when the function call is initiated before the current state is complete.

25. The medium as recited in claim 23, wherein said validating procedure includes:

sending a request including said Schema to a remote server; and

receiving a validity determination as to said Schema.

26. The medium as recited in claim 23, wherein said validating procedure includes:

reading said Schema into a set of valid descriptor classes; and

creating an instance of a compiler class wherein the compiler class is described in the Schema.

27. The medium as recited in claim 23, wherein the set of object oriented classes comprises code selected from the group consisting of: Java, C ++, and Smalltalk.

28. The medium as recited in claim 23, wherein said creating an instance procedure includes representing a naming space with the internal static class to provide an implementation of said structural complexity.

29. An apparatus for mapping a descriptive language including a data description having a structure complexity into an object oriented programming language, comprising:

means for receiving the data description;

means for identifying a complex-type element in the data description; and

means for creating an executable object oriented class that is independently executable in any of a plurality of run-time environments and that corresponds to the identified complex-type element, wherein the class includes an internal static class, wherein the internal static class corresponds to the structure complexity of the data description.

30. The apparatus as recited in claim 29, wherein said data description comprises an XML Schema.

31. The apparatus as recited in claim 29, further comprising means for validating a data description.

32. The apparatus as recited in claim 29, wherein said validating means further includes a object finite state machine including a current state to verify a mutator method call

against the current state of the object, wherein the data description is invalid when the mutator method call is initiated before the current state is complete.

33. The apparatus as recited in claim 29, wherein said validating means includes a web browser operable to send a request including said data description to a remote server for validation.

34. The apparatus as recited in claim 29, wherein said validating means includes:

means for reading said data description into a set of valid descriptor classes; and

means for creating a set of objects out of the data description wherein the occurrence of an object reflects validity.

35. The apparatus as recited in claim 29, wherein said object oriented classes comprise code selected from the group consisting of: Java, C++, and Smalltalk.

36. The apparatus as recited in claim 29, wherein said creating means includes means for representing a naming space with said internal static class to provide an implementation of said structure complexity.

IX. EVIDENCE APPENDIX

No Evidence Appendix is present in this appeal.

X. RELATED PROCEEDINGS APPENDIX

No Related Proceedings Appendix is present in this appeal.